Chapter 4
File Systems

File Systems

The best way to store information:

- Store all information in virtual memory address space
- Use ordinary memory read/write to access information
- Not feasible: no enough address space
- When process exits, information is lost

Another approach:

- Define named objects called files which hold programs and data
- Files are not part of the address space
- O.S. provides special operations (system calls) to create/destroy files
- Files are permanent records

File Systems

Essential requirements for long-term information storage:

- It must be possible to store a very large amount of information.
- The information must survive the termination of the process using it.
- Multiple processes must be able to access the information concurrently.

File Systems

Think of a disk as a linear sequence of fixed-size blocks and supporting reading and writing of blocks. Questions that quickly arise:

- How do you find information?
- How do you keep one user from reading another's data?
- How do you know which blocks are free?

File Naming

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Figure 4-1. Some typical file extensions.

User’s View of the File System

A file system consists of two distinct parts

1. A collection of files, each storing related data
2. A directory structure, which organizes and provides information about all the files in the system

Three common file organizations

1. Unstructured byte sequence
The basic unit for reading and writing is byte, sequential access.

Example: Unix.

User’s View of the File System

2. Sequence of fixed-size records
The basic unit is a record, sequential access. Records can be read and written, but cannot be inserted or deleted in the middle of a file.

Example: CP/M.

3. A tree of disk blocks
The basic unit is a record. Each block holds n keyed records. Records can be looked up by key and inserted. Blocks can be split.

Used on mainframe computers.

File Structure

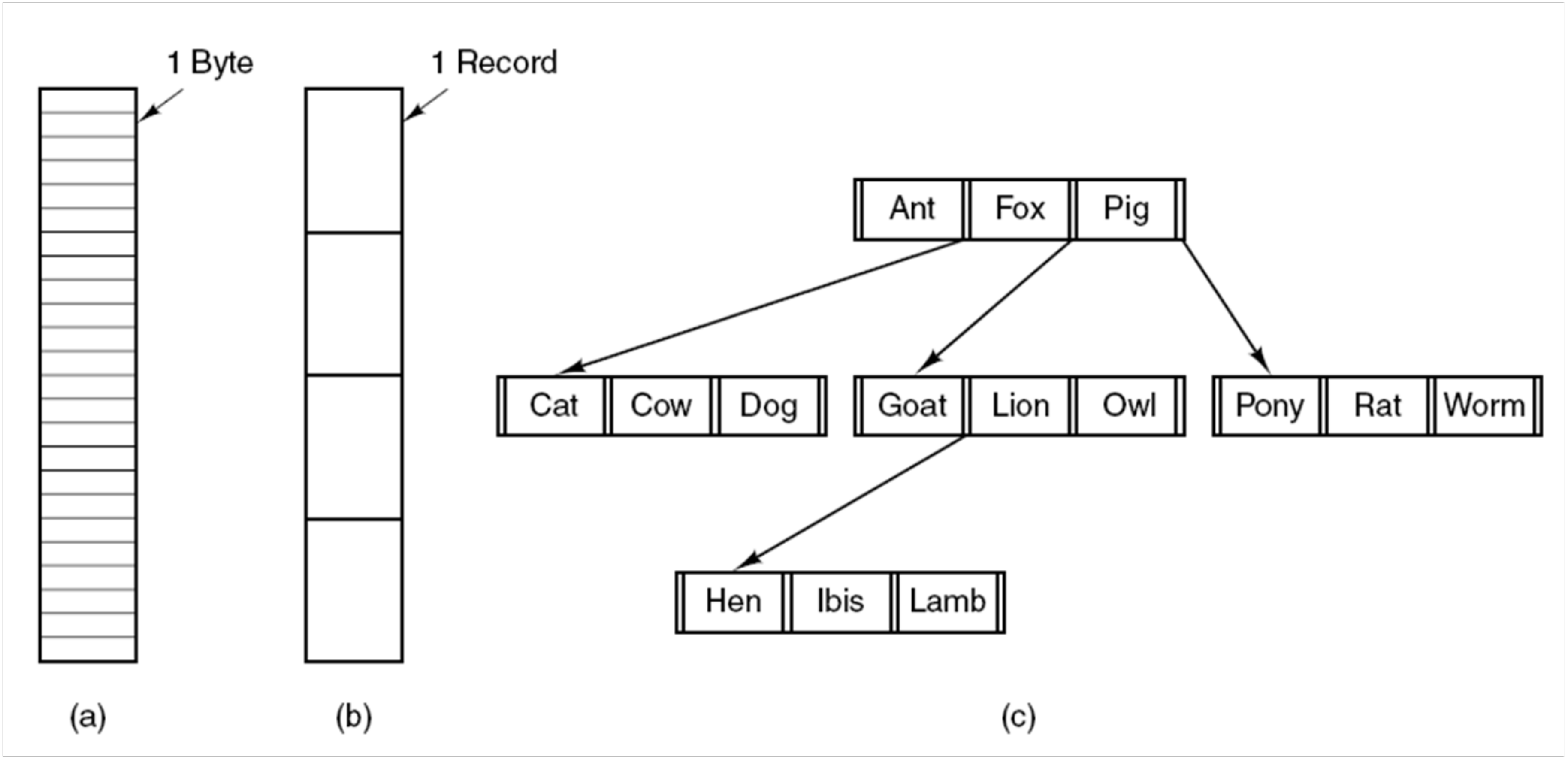


Figure 4-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

File Types

Regular files: contain data bytes

Directories: contain the names of other files

Special files: I/O devices

- Character special files: e.g. terminals, printers
- Block special files: e.g. disks, tapes

File access methods

- Sequential access: start at the beginning, read the bytes or records in order
- Random access: read bytes or records in any order

File Types

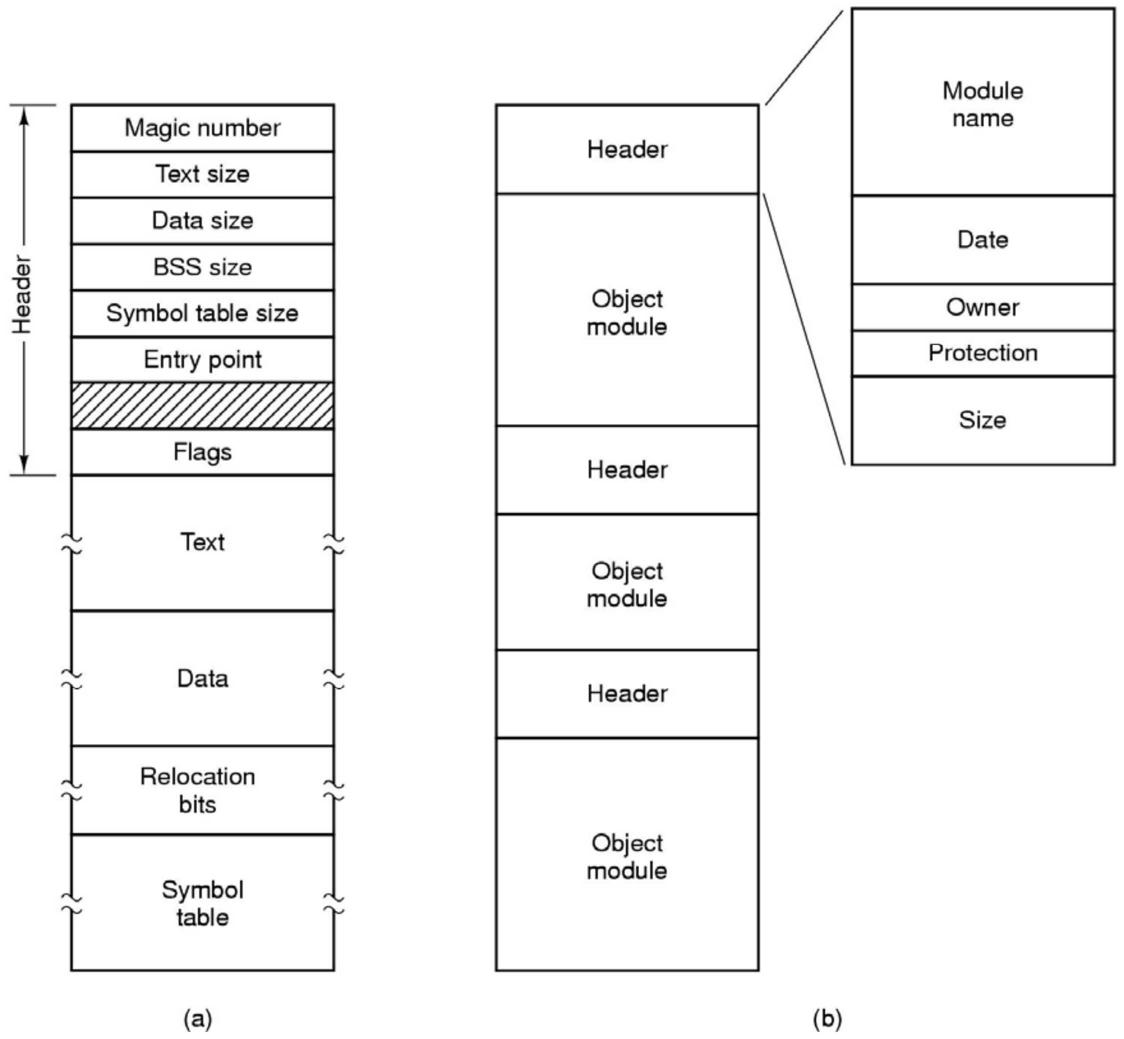
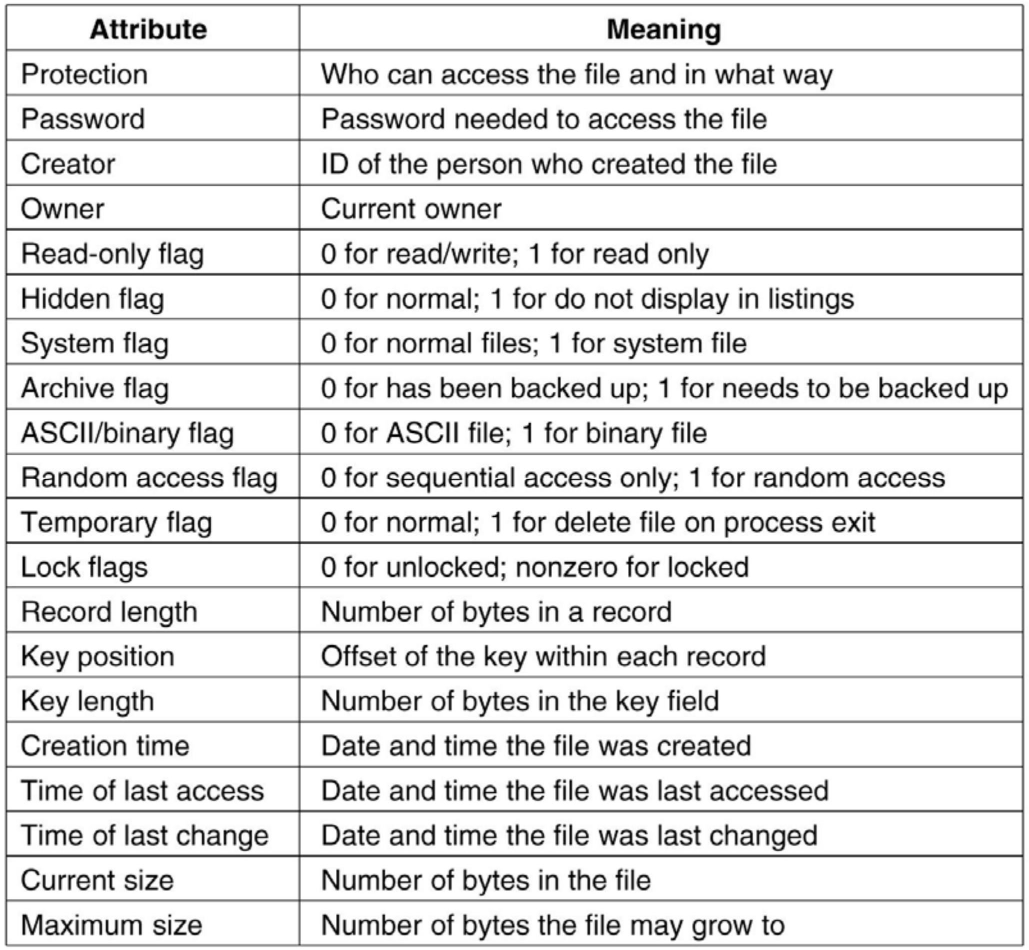


Figure 4-3. (a) An executable file. (b) An archive.

File Attributes

File attributes: Extra information associated with each file



Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Figure 4-4a. Some possible file attributes.

File Operations

The most common system calls relating to files:

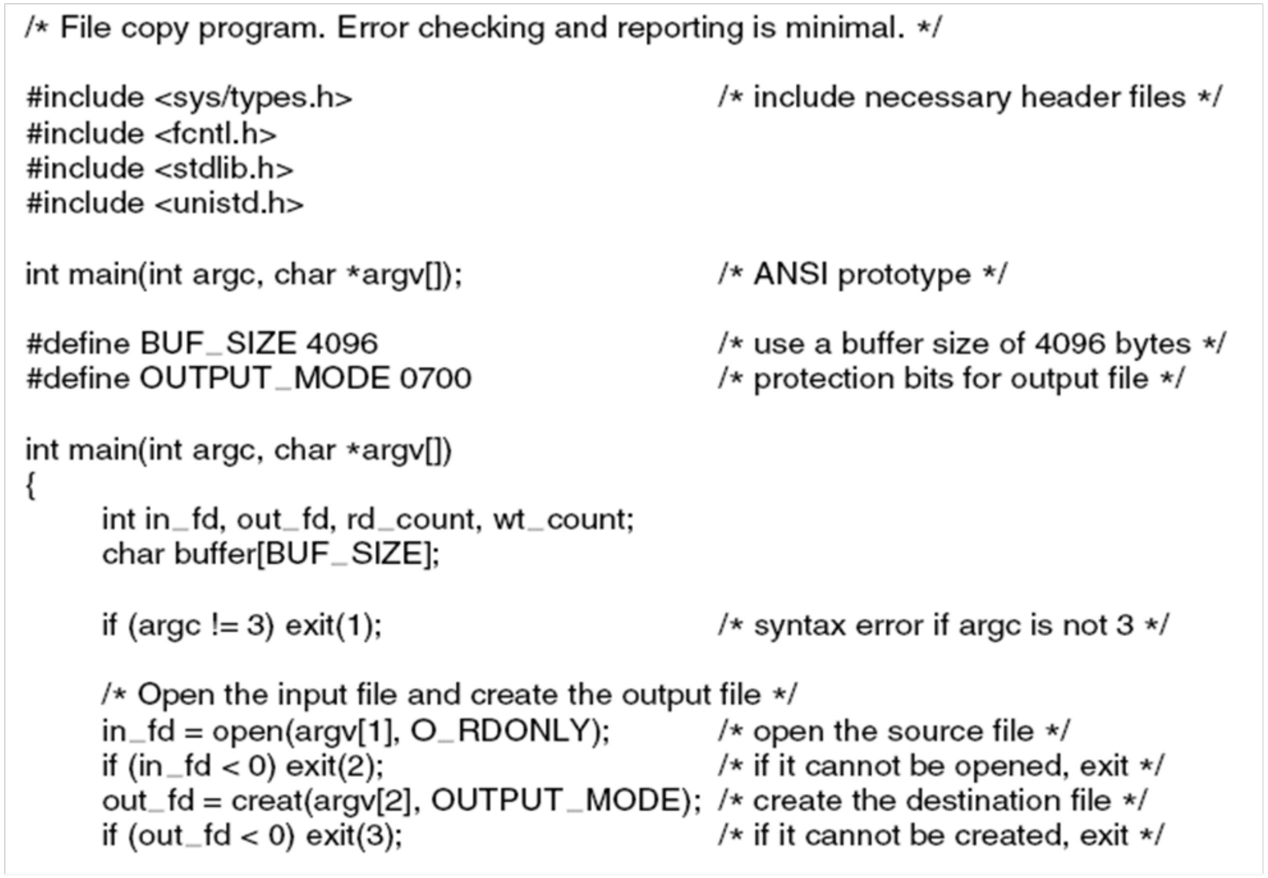
- Create
- Delete
- Open
- Close
- Read
- Write
- Append
- Seek
- Get Attributes
- Set Attributes
- Rename

Open: bring attributes and disk addresses to memory for fast access

Close: put back to disk

Seek: put the pointer to a specific location in the file

Example Program Using File System Calls (1)



```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                       /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);            /* ANSI prototype */

#define BUF_SIZE 4096                        /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                     /* protection bits for output file */

int main(int argc, char *argv[])
{
     int in_fd, out_fd, rd_count, wt_count;
     char buffer[BUF_SIZE];

     if (argc != 3) exit(1);                 /* syntax error if argc is not 3 */

     /* Open the input file and create the output file */
     in_fd = open(argv[1], O_RDONLY);        /* open the source file */
     if (in_fd < 0) exit(2);                 /* if it cannot be opened, exit */
     out_fd = creat(argv[2], OUTPUT_MODE);   /* create the destination file */
     if (out_fd < 0) exit(3);                /* if it cannot be created, exit */
```

. . .

Figure 4-5. A simple program to copy a file.

Example Program Using File System Calls (2)

```
        /* Copy loop */
        while (TRUE) {
             rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
        if (rd_count <= 0) break;                 /* if end of file or error, exit loop */
             wt_count = write(out_fd, buffer, rd_count); /* write data */
             if (wt_count <= 0) exit(4);          /* wt_count <= 0 is an error */
        }

        /* Close the files */
        close(in_fd);
        close(out_fd);
        if (rd_count == 0)                        /* no error on last read */
             exit(0);
        else
             exit(5);                             /* error on last read */
    }
```

Figure 4-5. A simple program to copy a file.

Directories

1. Single directory shared by all users
   Different users cannot use the same file name.

2. One directory per user
   Different users can use the file name. But each user can keep only a sequence of files, no any group support.

3. Arbitrary tree per user
   Flexible. Used in Unix.

Hierarchical Directory Systems (1)

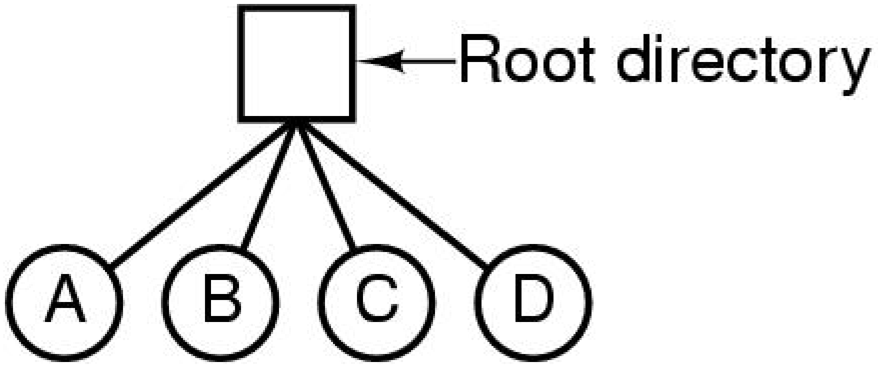


Figure 4-6. A single-level directory system containing four files.

Hierarchical Directory Systems (2)

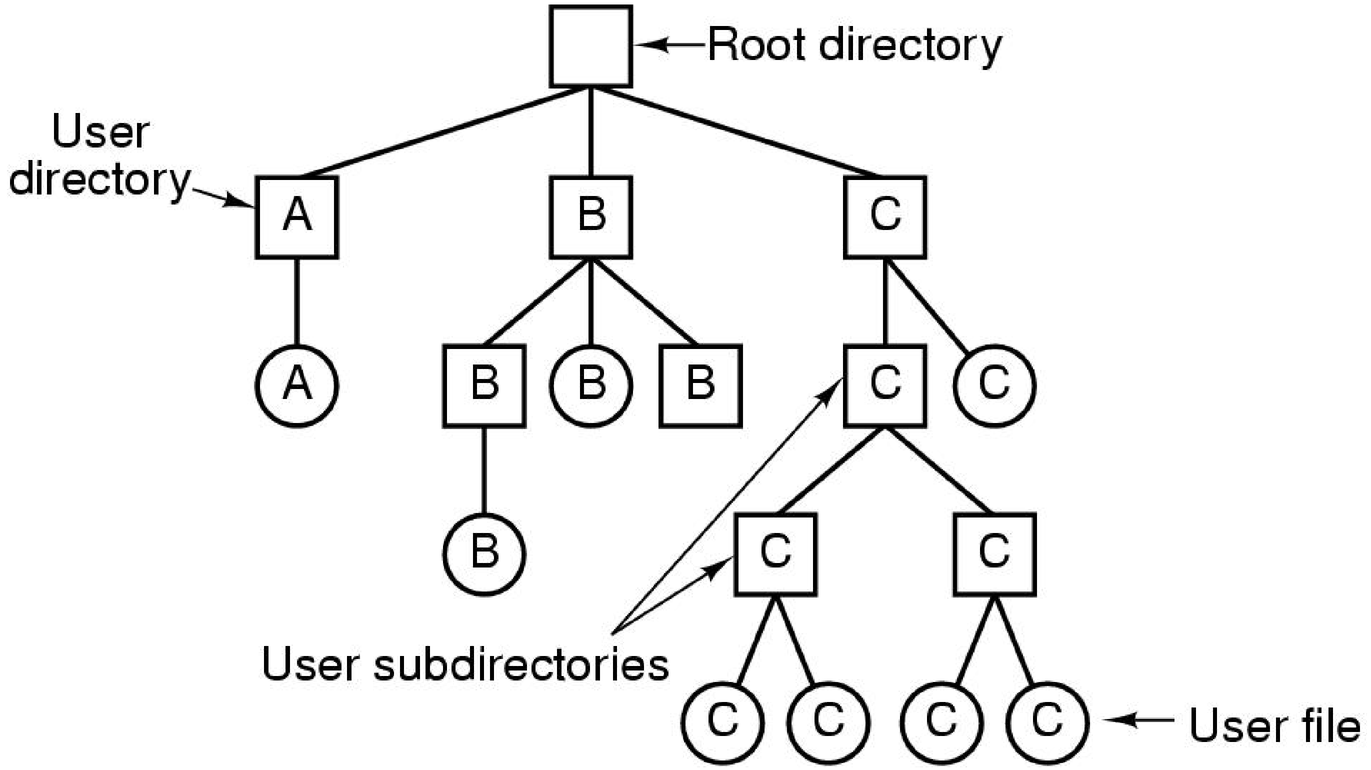


Figure 4-7. A hierarchical directory system.

Path Names

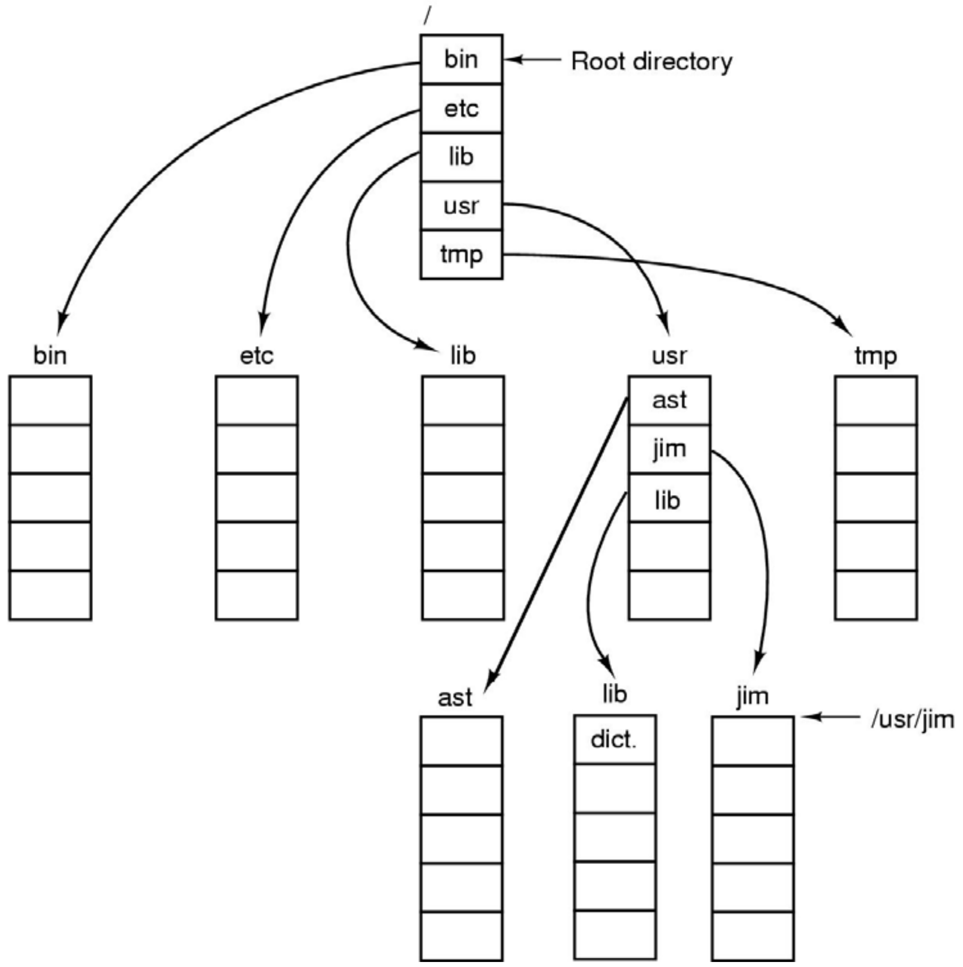


Figure 4-8. A UNIX directory tree.

Directory Operations

System calls for managing directories:

- Create
- Delete
- Opendir
- Closedir
- Readdir
- Rename
- Link
- Uplink

File System Implementation (Designer’s View)

How to store n bytes on disk

1.Allocate n consecutive bytes of disk space

For insertion, have to move the file (slow)

2. The file is split up into a number of (not necessarily contiguous) fixed size blocks

More expensive to keep track of where the blocks are.

Block Size

Candidates: sector, track, cylinder

The time to read a block from disk:
Seek time + rotation time + transfer time

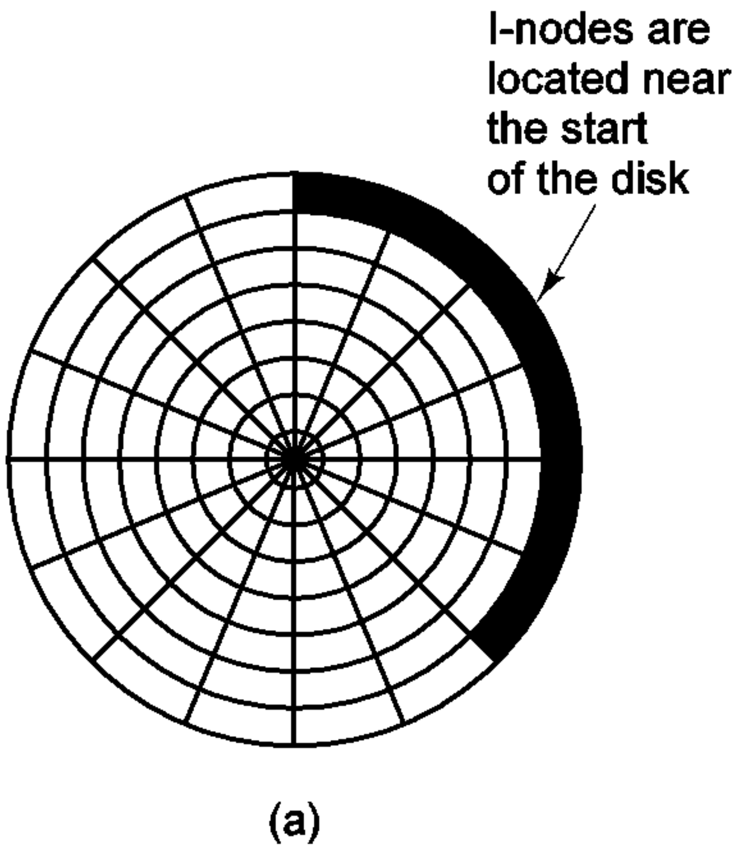
Example:
A disk with 32768 bytes per track
Rotation time: 16.67 msec
Average seek time 30 msec

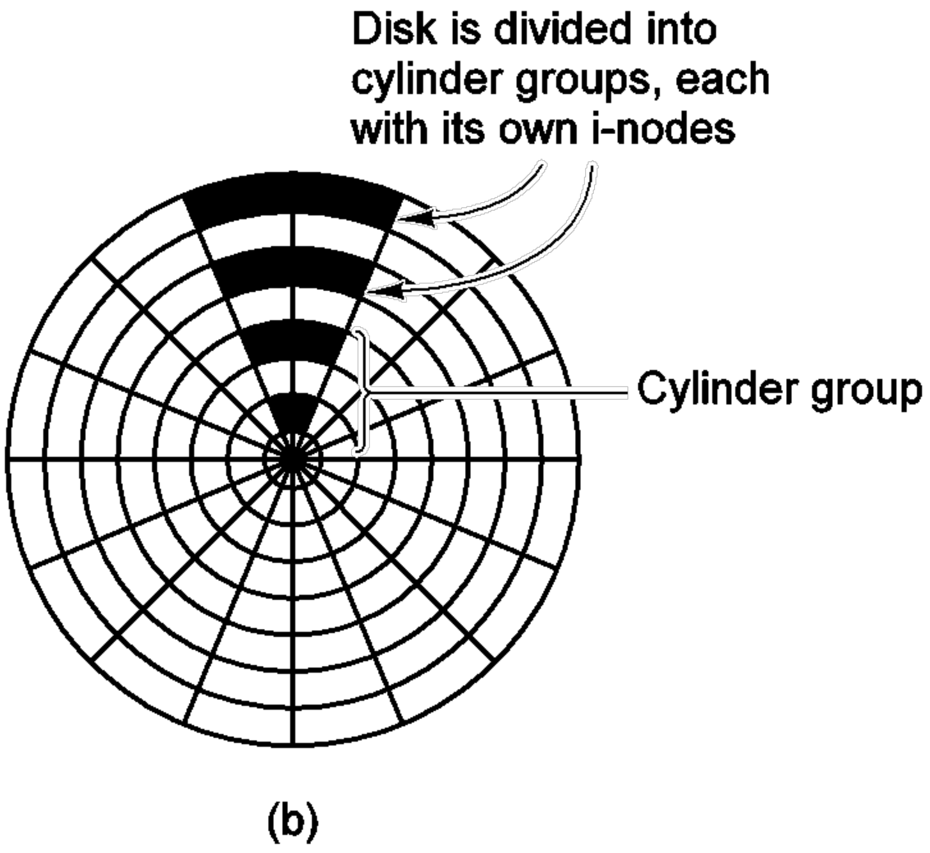
Block size K bytes
The time to read a block:
30 + 16.67/2 + (K/32768) x16.67

Large block size: poor disk space utilization
Small block size: reading a block is slow

Compromise between the speed and utilization:
General block size: 512, 1K or 2K

Block Size


I-nodes are
located near
the start
of the disk
(a)


Disk is divided into
cylinder groups, each
with its own i-nodes
Cylinder group
(b)

Disk Space Management Block Size

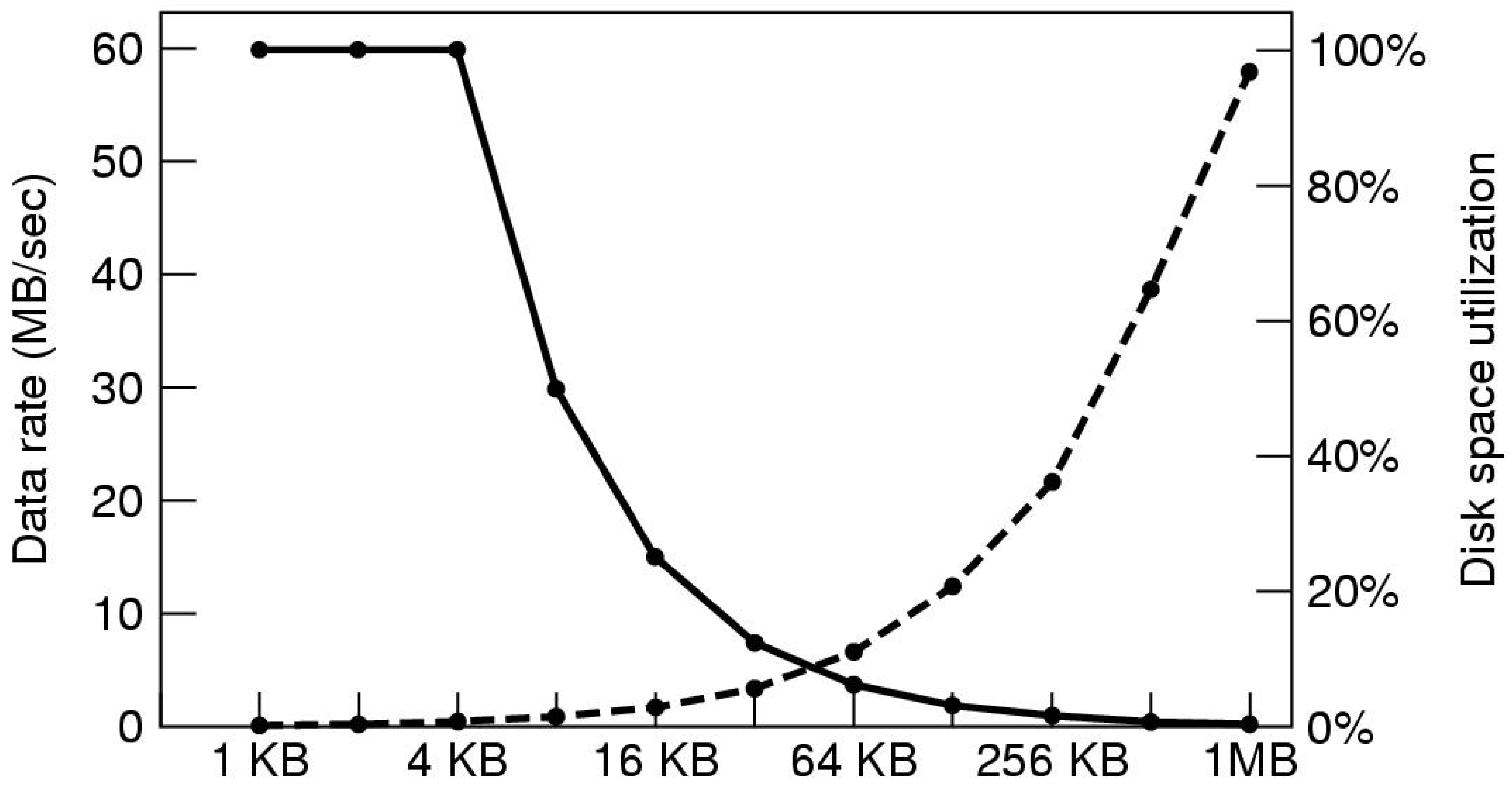


Figure 4-21. The solid curve (left-hand scale) gives the data rate of a disk. The dashed curve (right-hand scale) gives the disk space efficiency. All files are 4 KB.

Keeping Track of Blocks of Each File

1. Store blocks consecutively

   When file grows, may cause problem.

2. Linked list

   Use the last two bytes as s pointer to point to the next block in the file.

   Problems:

   a. The number of bytes in a block is no longer power of 2

   b. Random access is slow

Contiguous Allocation

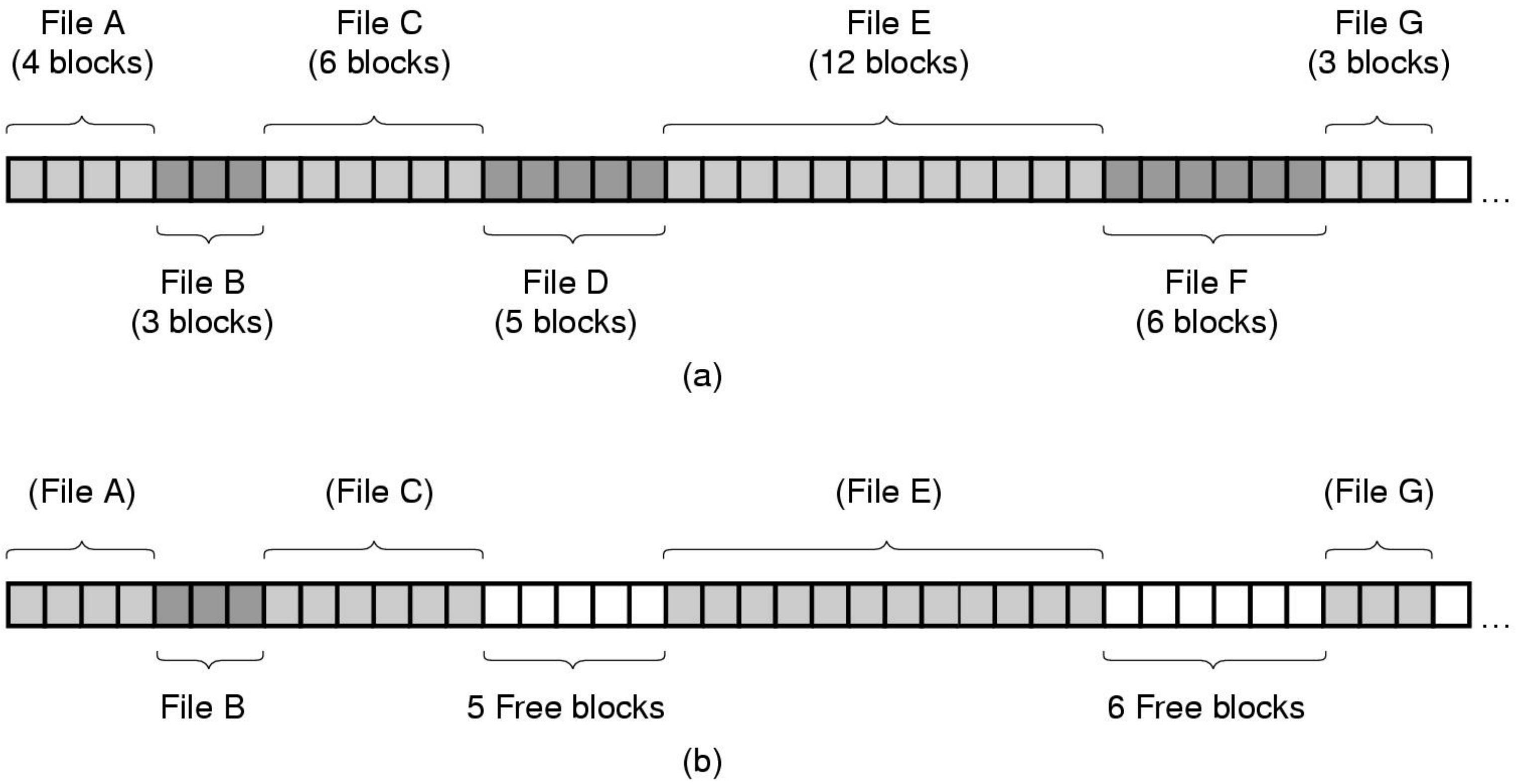


Figure 4-10. (a) Contiguous allocation of disk space for 7 files. (b) The state of the disk after files D and F have been removed.

Linked List Allocation

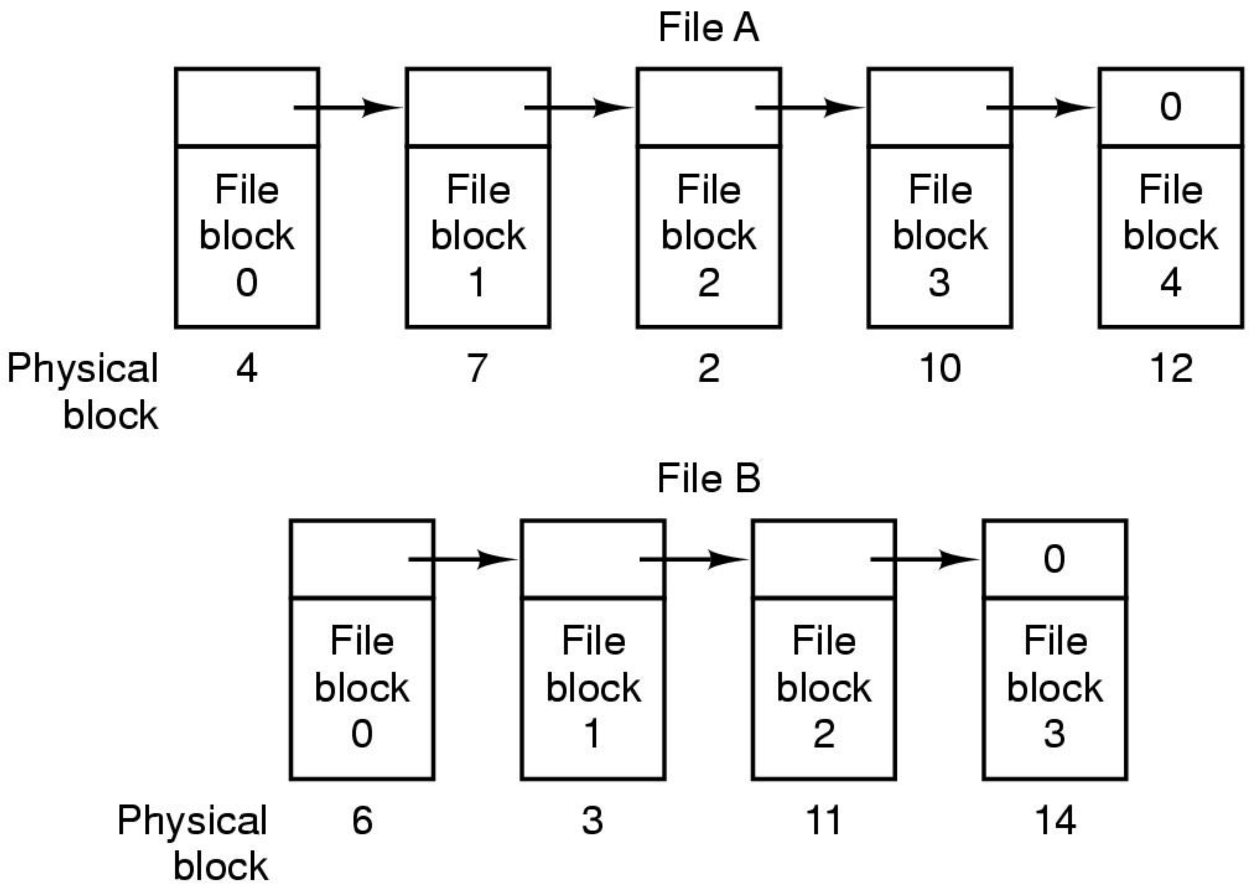


Figure 4-11. Storing a file as a linked list of disk blocks.

Linked List Allocation Using a Table in Memory

File allocation table (FAT)

A table has one entry for each disk block. All blocks of a file linked together.

- The directory keeps the first block number of each file.
- The table is stored in memory.
- Example system: MS-DOS
- Problem:

  Pointers for all files on the disk are mixed up in the same table. An entire FAT is potentially needed for accessing one file.

Linked List Allocation Using a Table in Memory

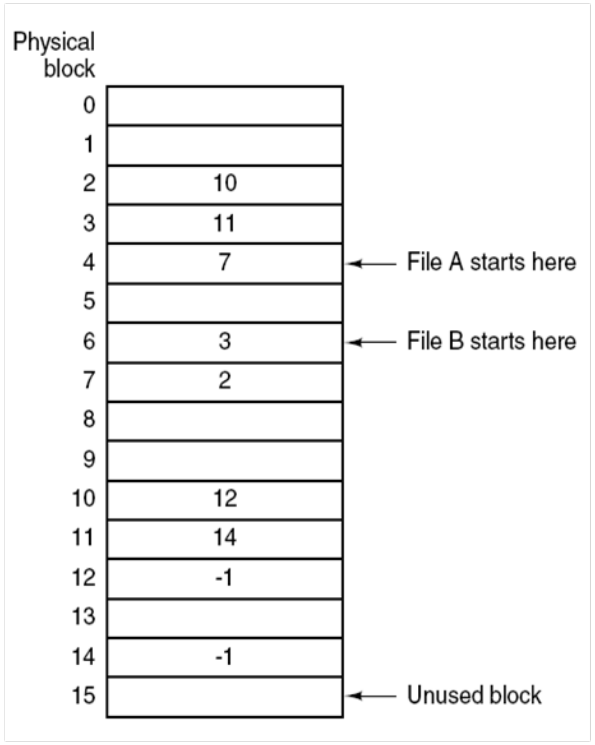


Figure 4-12. Linked list allocation using a file allocation table in main memory.

I-nodes

Keep the block lists for different files in different places

I-node (index node) in Unix.

Each file has a small table called I-node
I-node contains attributes, first 10 blocks of the file, and 3 indirect block numbers.

Single indirect block: points to data blocks
Double indirect block: points to single indirect blocks
Triple indirect block: points to double indirect blocks

I-nodes

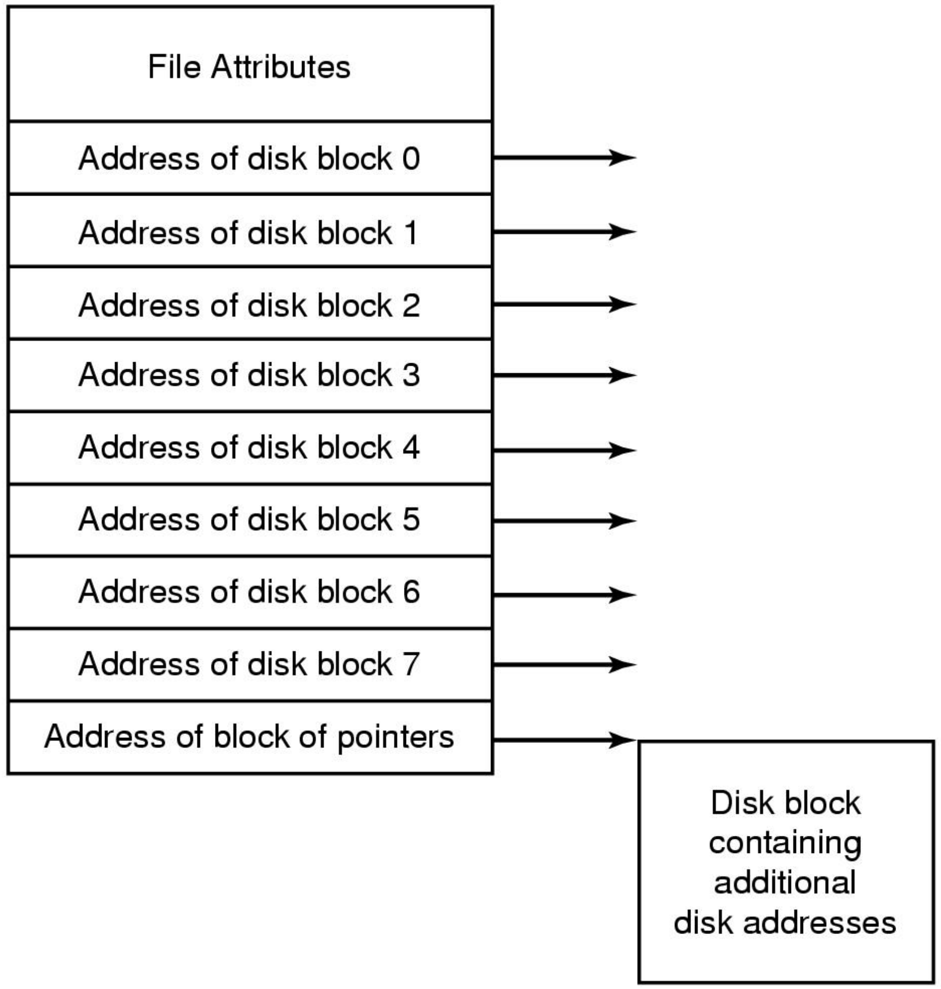


Figure 4-13. An example i-node.

I-nodes

Example:
For 1K disk block and 32-bit disk address, how large a file could be?

•Single indirect block holds 256 disk addresses
•Double indirect block holds 256 single indirect blocks

•File size < 10 blocks: no indirect blocks
•File size between 10 and 266 blocks: single indirect
•File size between 267 and 266 + 256 x 256 = 65802K blocks, use double indirect
•File size between 64M and 16G, triple indirect
•File size > 16 G, cannot handle

I-nodes

Advantage: fast random access

At most three disk references are needed to locate the disk address of any byte in the file.

When the file is open, its I-node is brought into memory until the file is closed.

Example of finding the 300th block in a file:
300 - 266 = 34
Follow double indirect to the first single indirect, the 34th block.
No need to follow the 299 blocks before this block as in the linked list case.

Implementing Directories

Directory organization

Single directory in the system.
Example: CP/M

Hierarchical directory tree.

Example 1: MS-DOS
Attributes: 8 1-bit flags: b7, b6, …, b0
b0 = 1: read only file
b1 = 1: hidden file
b2 = 1: system file
b4 = 1: a subdirectory

Example 2: Unix

Implementing Directories (1)

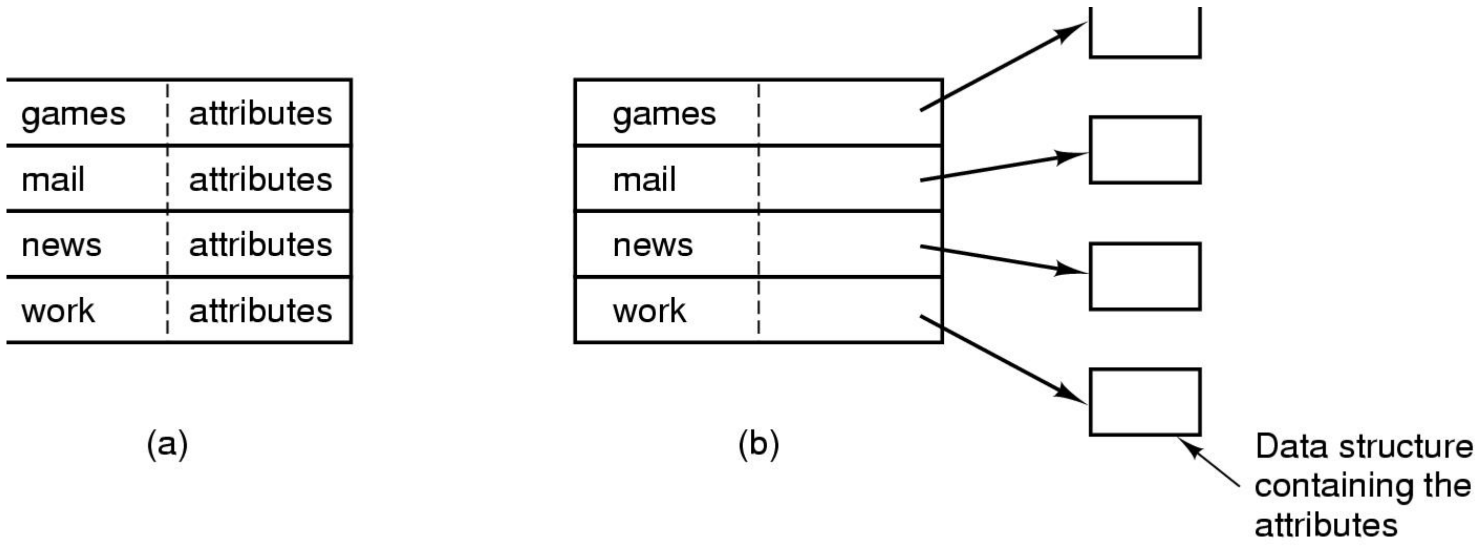


Figure 4-14. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

The MS-DOS File System (1)

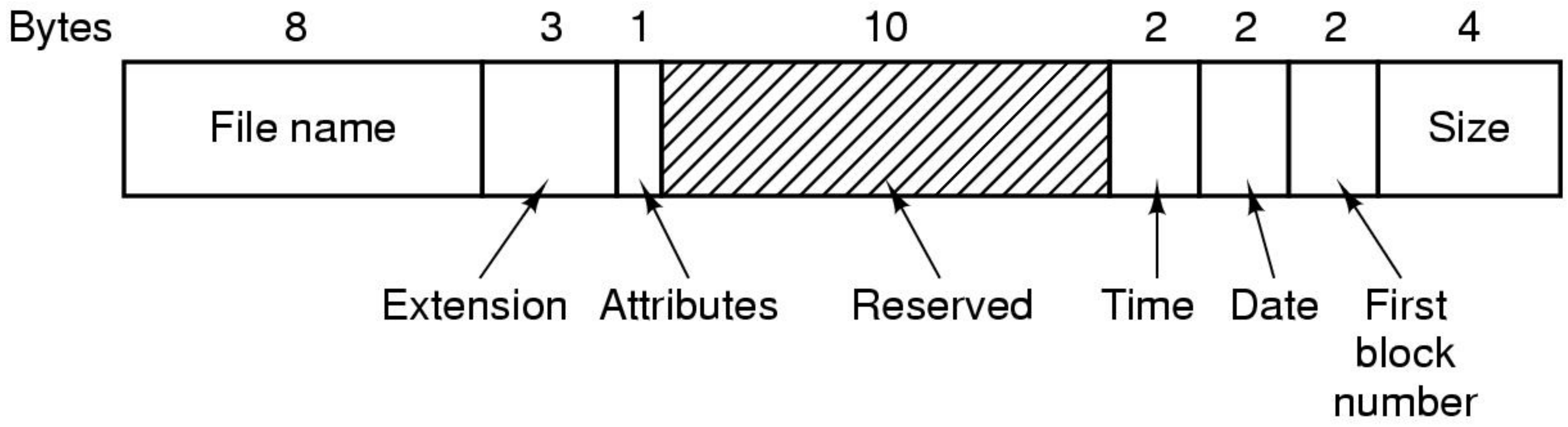


Figure 4-31. The MS-DOS directory entry.

The UNIX V7 File System (1)

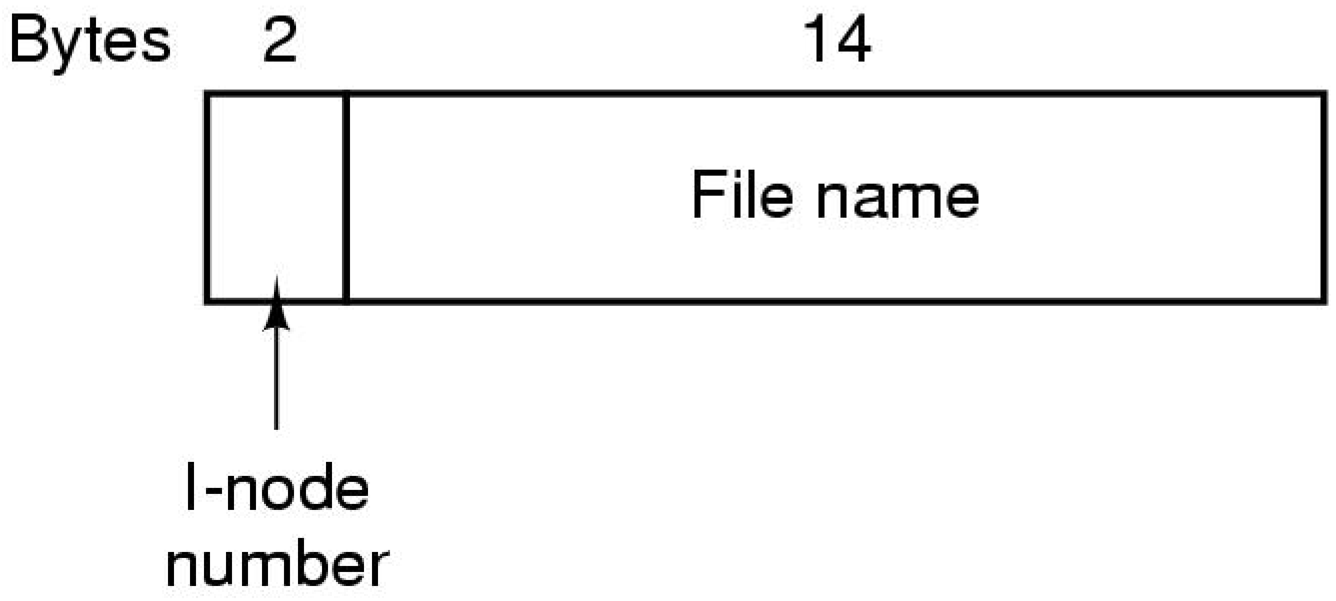


Figure 4-33. A UNIX V7 directory entry.

The UNIX V7 File System (2)

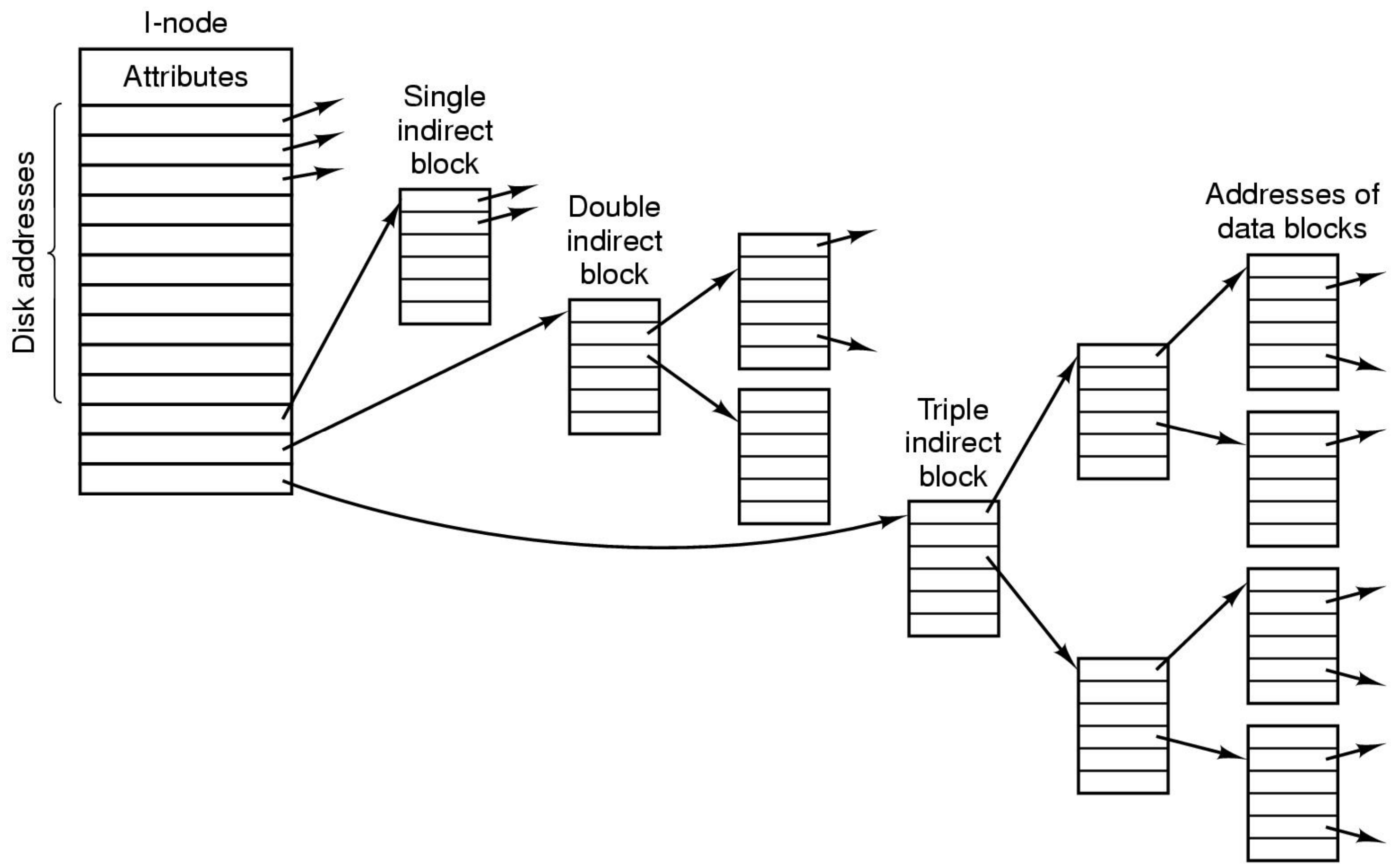


Figure 4-34. A UNIX i-node.

The UNIX V7 File System (3)

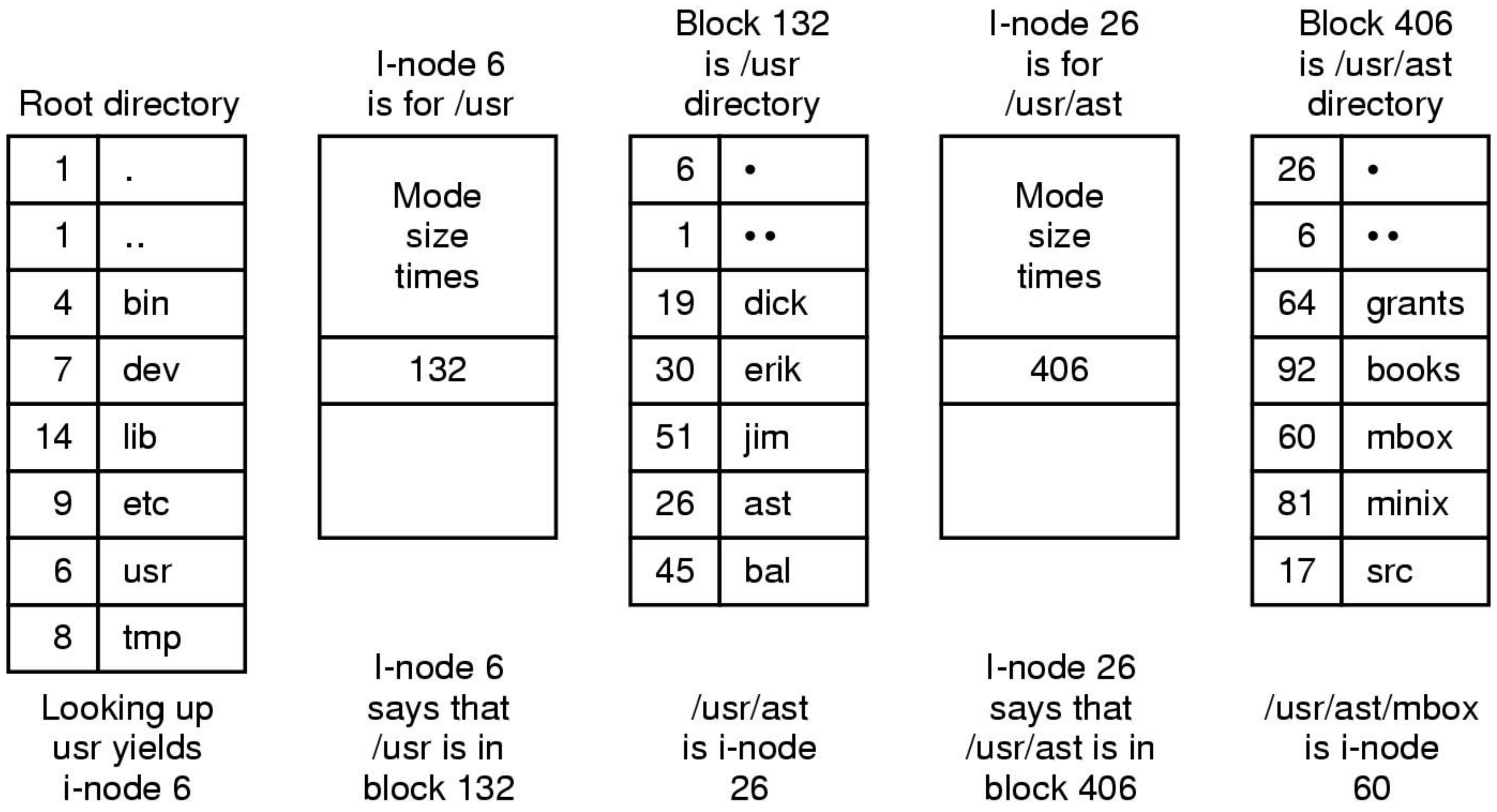

Root directory
1 .
1 ..
4 bin
7 dev
14 lib
9 etc
6 usr
8 tmp
Looking up usr yields i-node 6
I-node 6 is for /usr
Mode size times
132
I-node 6 says that /usr is in block 132
Block 132 is /usr directory
6 •
1 ••
19 dick
30 erik
51 jim
26 ast
45 bal
/usr/ast is i-node 26
I-node 26 is for /usr/ast
Mode size times
406
I-node 26 says that /usr/ast is in block 406
Block 406 is /usr/ast directory
26 •
6 ••
64 grants
92 books
60 mbox
81 minix
17 src
/usr/ast/mbox is i-node 60

Figure 4-35. The steps in looking up */usr/ast/mbox*.

Shared Files

A shared file appears simultaneously in different directories belonging to different users

File system is a directed acyclic graph, not a tree

Problems: When directories contain disk addresses (e.g. CP/M), cannot share files.

Solution: Directory does not point to disk addresses. Point to a data structure associated with the file. Use Link Count to record how many users are sharing the file.

When performing ``remove file'', only when count = 0, remove the file, otherwise reduce the Link Count.

Shared Files

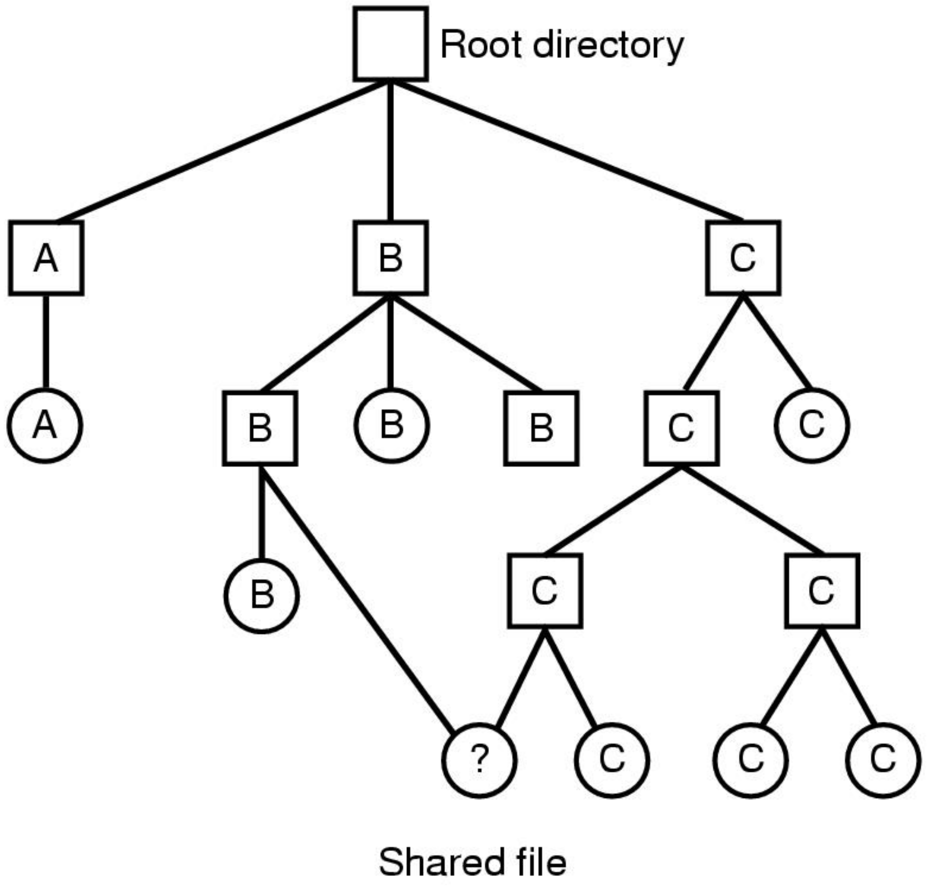


Figure 4-16. File system containing a shared file.

Shared Files

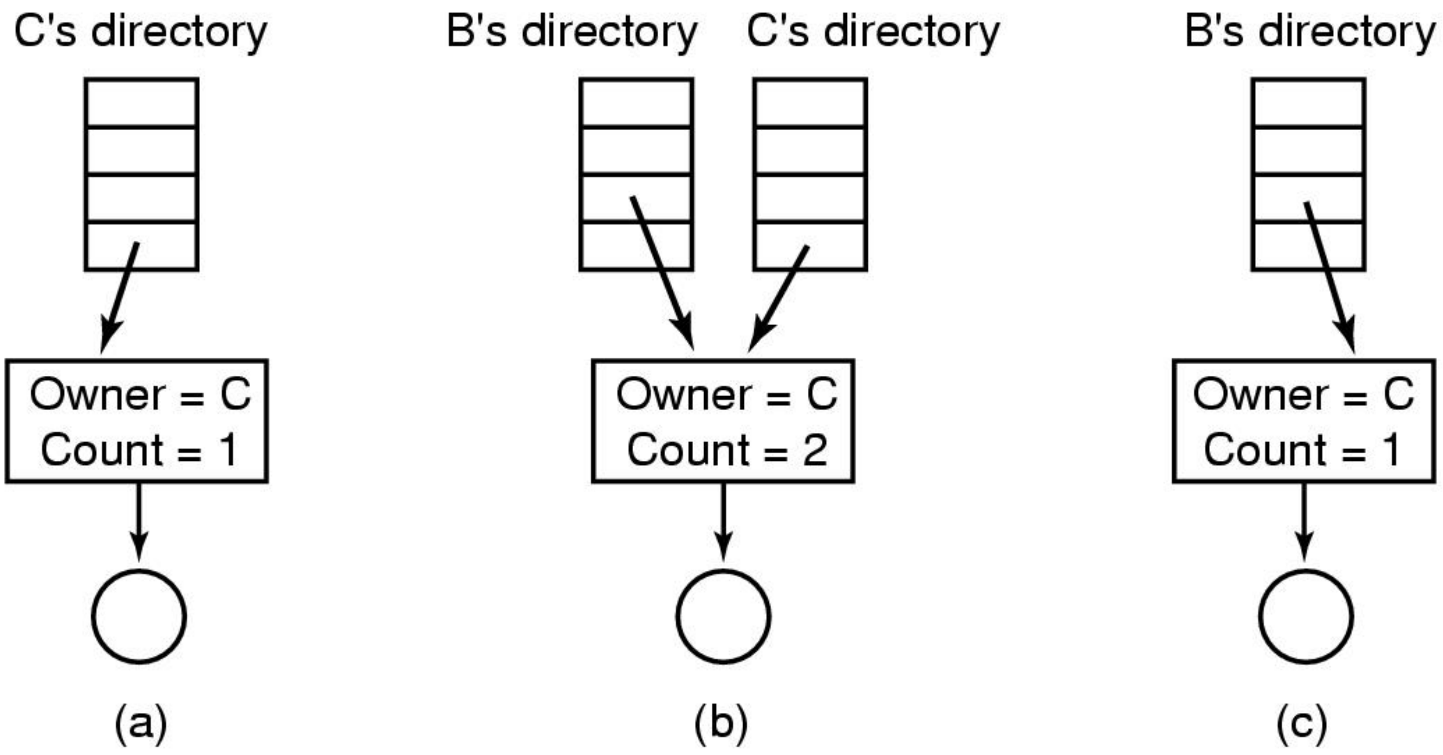


Figure 4-17. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

Shared Files

Example: Unix. To share a file, different directories point to the same i-node.

Problem with pointing to the same data structure:
After a file owner removes the file, counting information may still be charged to the file owner.

Solution: use symbolic linking:
Create a new file, of type LINK. Enter that file in another user's directory. The new file only contains the path to the shared file. Can be used on different machines.

Problem with symbolic linking:
Extra overhead to access file: read extra block to get the i-node.

Shared Files

Unix provides both ways

ln file1 file 2
Points to the same i-node. Can be seen by
ls -i file1
ls -i file2

ln -s file link
Create a symbolic link: link

ls -l will show
l - - - - - - link@! file

more link is equivalent to **more file**

Journaling File Systems

Operations required to remove a file in UNIX:

- Remove the file from its directory.
- Release the i-node to the pool of free i-nodes.
- Return all the disk blocks to the pool of free disk blocks.

Keeping Track of Free Blocks

1. Linked list

   Use several blocks to store free block numbers. Link these blocks together. Search is fast.

   Example: 1K block size. 16-bit disk block number (20 M disk). Need 40 blocks to hold all 20K block numbers.

2. Bit map

   n blocks require n bits. Smaller space.
   Search is slow. May be put in memory.

   Example: 20 M disk requires only 20K bits (3 blocks).

Keeping Track of Free Blocks

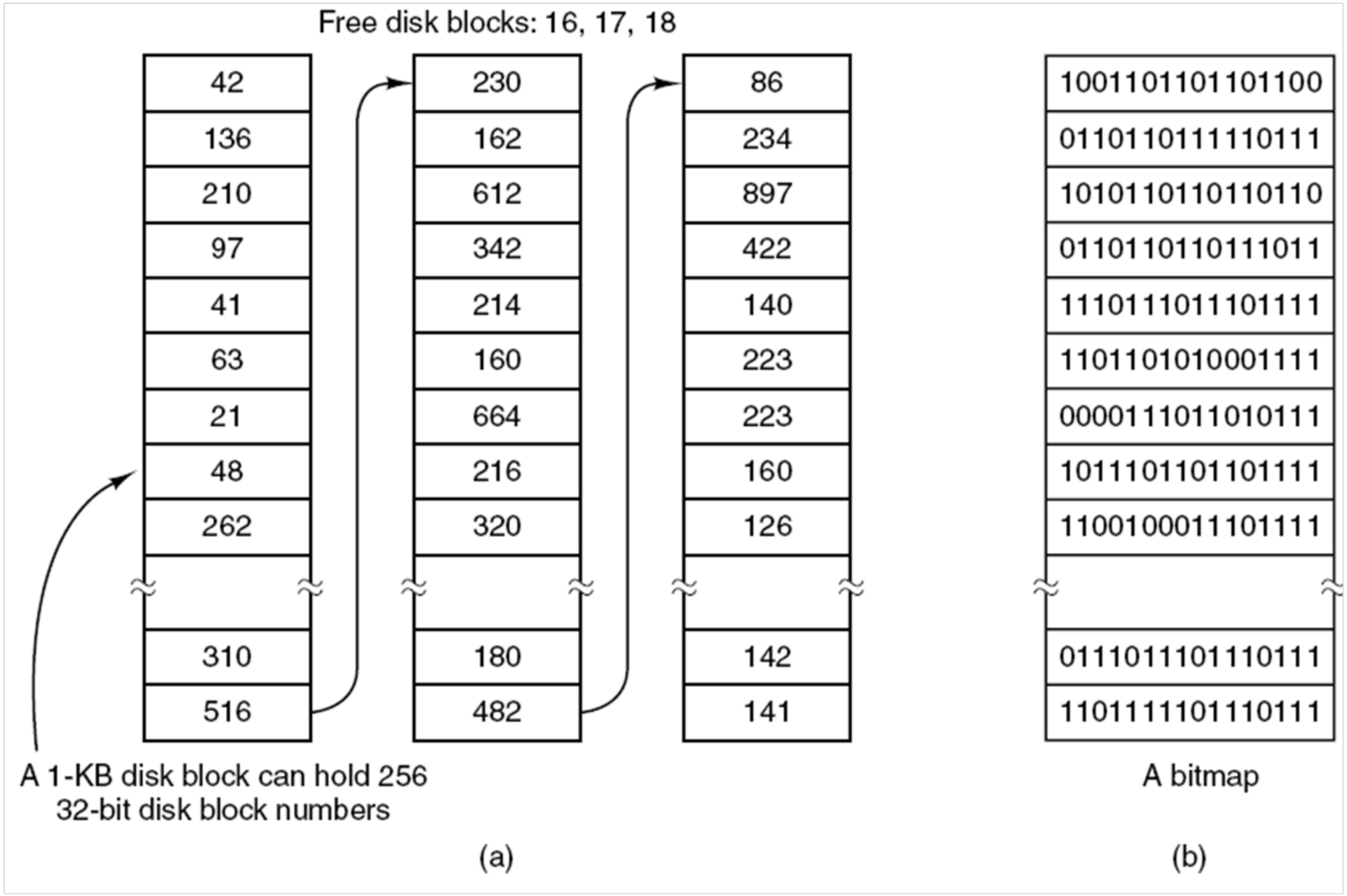


Figure 4-22. (a) Storing the free list on a linked list. (b) A bitmap.

File System Consistency

After system crashes, the file system may be in an inconsistent state.

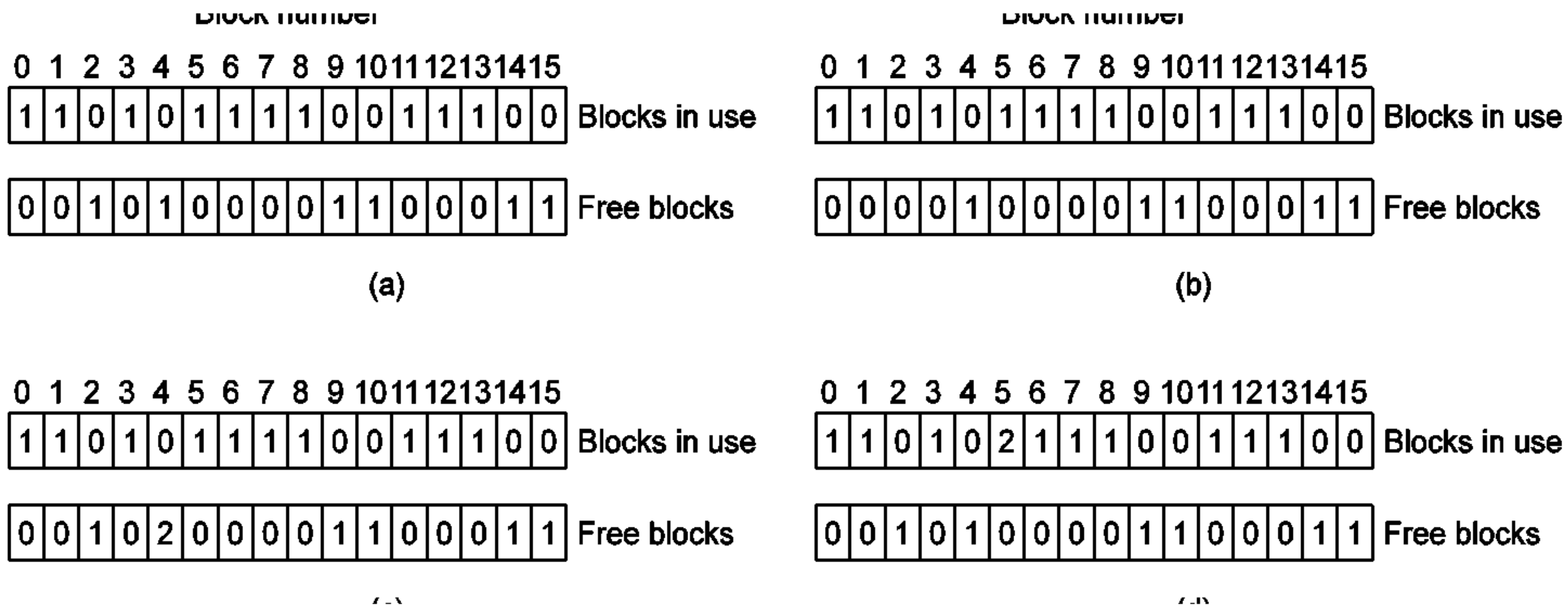
A utility program checks:

1.Block consistency: a block is either free or in a file.

Two counters per block: one counter counts the times the block presents in a file; another counter counts the times the block presents in the free list.

Read all I-nodes: for each block number in I-node, increment the first counter
Check the free list, increment the second counter
Consistent: counter1/counter2 = 1/0 or 0/1
Missing block: 0/0
Correction: put the block in free list
Duplicated block in free list: 0/2
Correction: delete one from free list
Duplicated data block: 2/0
Correction: allocate a free block, copy the data of the block into it and insert the copy to one of the file
1/1: should be corrected to 1/0

File System Consistency



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

(a)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

(b)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1	Free blocks

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	2	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

Figure 4-27. File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

File System Consistency

2. File consistency: check how many users are sharing a file

One counter per file (per I-node)
Starts from root directory, inspect all files. For each file, increment the counter for the file's I-node
Compare the counters with the Link Count in the I-node
_ Consistent: Link Count = counter
_ Inconsistent: Link Count not = counter
_ Link Count > counter: all files removed, I-node is still not removed, waste space
_ Link Count < counter: file may point to a released I-node
Solution: Force the Link Count = actual number of directory entries

3. Other heuristic checks
I-node number > it should be
Strange mode, e.g., 777
Directories with too many entries directory.

File System Performance

Disk access is about 100,000 times slower than memory access

Reduce the number of disk accesses needed

Block cache: keep some blocks (logically belong to the disk) in memory

Read a block, first check cache. If the block is not in cache, read it into cache, then use it.

For full cache, replace one block
Replacement algorithm: similar to virtual memory (FIFO, LRU,...), but need to consider special features of files

File System Performance

Blocks may not have good locality

Which blocks will be needed again soon
Indirect blocks are seldom used again.
Partially full data block may be needed soon (being written).

Which blocks are essential to the consistency of the file system
I-node, indirect, and directory blocks are essential, and need to be written to disk as soon as modified.

Data blocks should not stay in cache for too long without writing them out (possible data loss when system break down).

Two ways to deal with this:

(1) Provide a system call which forces all modified blocks out into disk.
Unix system call SYNC, called every 30 sec.

(2) Write through cache. All modified blocks are written back to disk immediately.
Example: MS-DOS.

Caching

- Some blocks, such as i-node blocks, are rarely referenced two times within a short interval.

- Consider a modified LRU scheme, taking two factors into account:

  - Is the block likely to be needed again soon?
  - Is the block essential to the consistency of the file system?

Caching

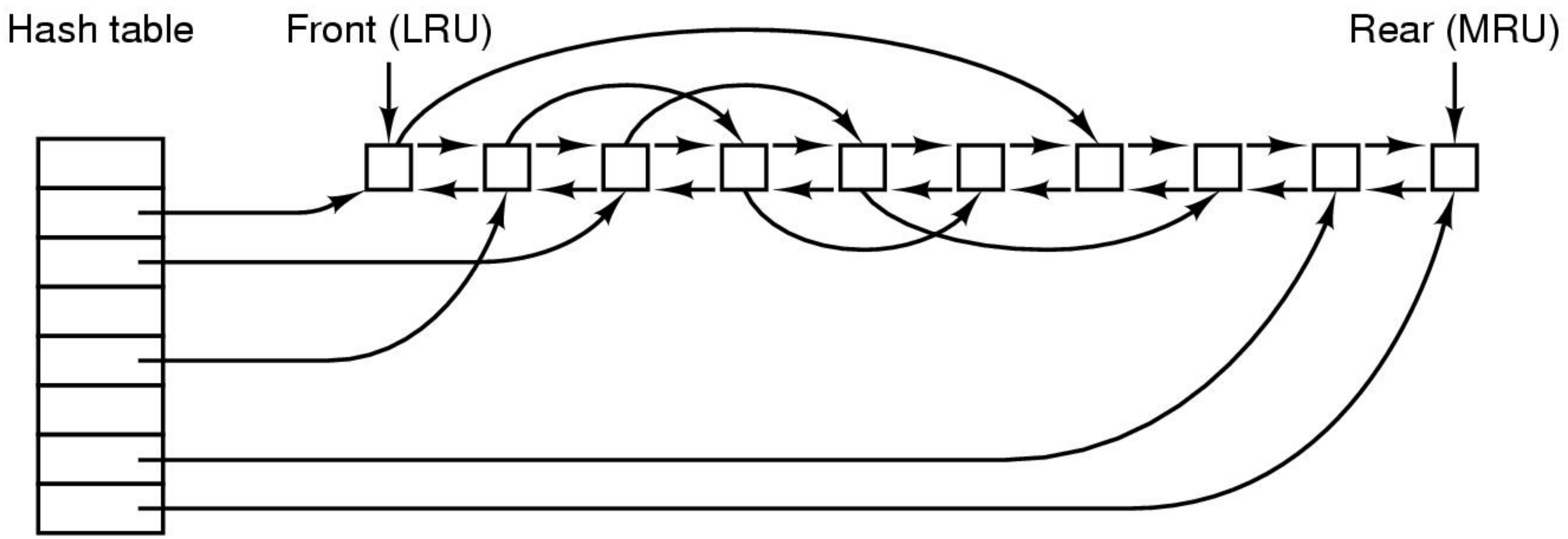


Figure 4-28. The buffer cache data structures.

Reducing Disk Arm Motion

1. Allocate the blocks that are likely to be accessed in sequence in the same cylinder.

How to get consecutive free blocks.
If linked list is used for free blocks, keep track of disk usage not in blocks, but in groups of consecutive blocks (e.g. in units of 2 blocks. Allocation unit is still a block, but 2 consecutive blocks in a file are consecutive on disk).

2. Reduce rotation time:
Place consecutive blocks in a file in the same cylinder, but interleaved for maximum throughput.

Example:
Rotation time = 16:67 ms
Block transfer time 4 ms
4 way interleave. One rotation can read 4 blocks

Reducing Disk Arm Motion

For the system using I-node

First access I-node then data block. Put I-node close to data block.

Usually I-node placed at the start of the disk. On the average, data block is ½ of the number of cylinders away.

Put I-node at middle of the disk. Average distance: 1/4 of the number of the cylinders.

Divide disk into cylinder groups. Each with its own I-nodes, blocks and free list. For an I-node in each group. Try to allocate blocks within the group.

Reducing Disk Arm Motion

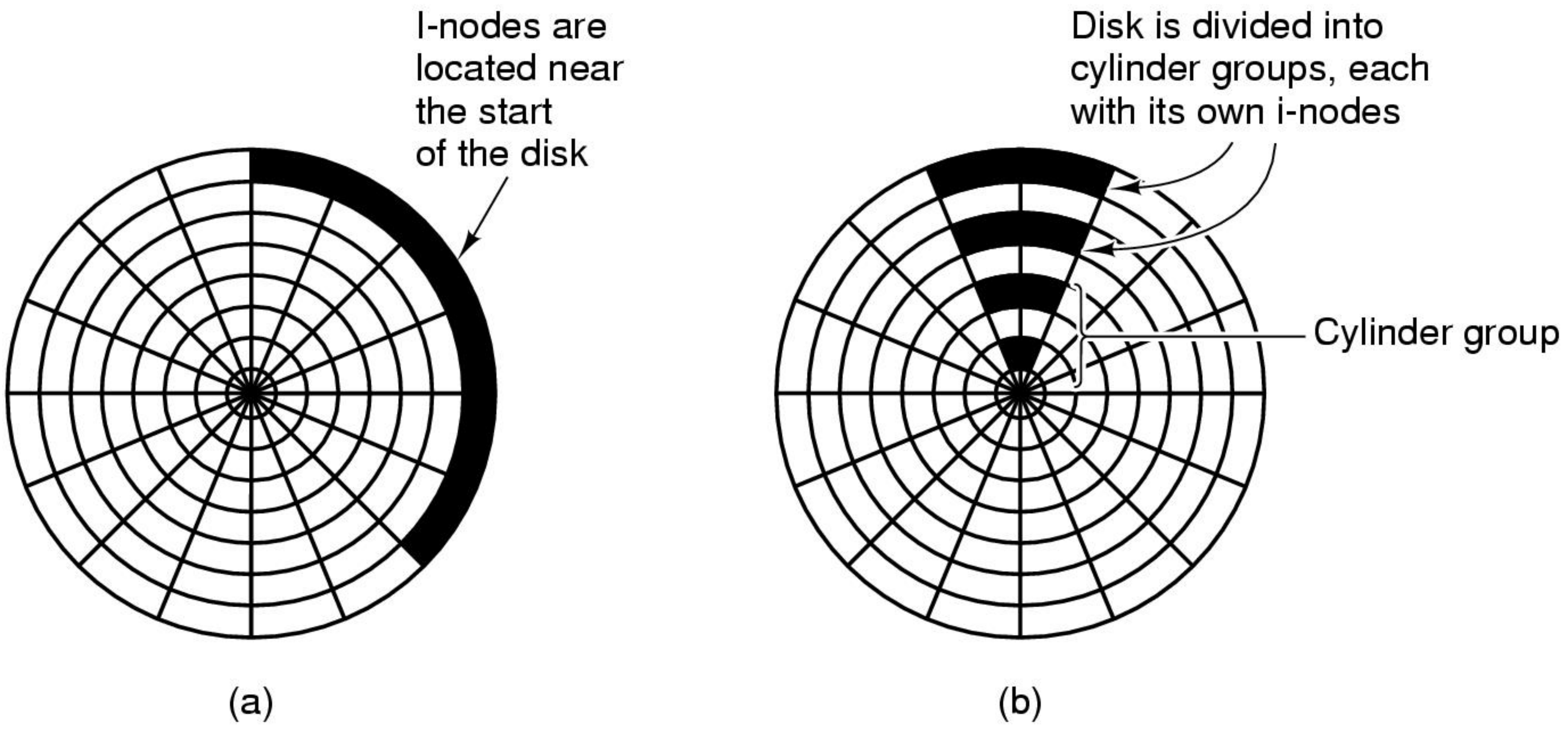

I-nodes are located near the start of the disk
(a)
Disk is divided into cylinder groups, each with its own i-nodes
Cylinder group
(b)

Figure 4-29. (a) I-nodes placed at the start of the disk. (b) Disk divided into cylinder groups, each with its own blocks and i-nodes.